# Proposed Pseudorandom Number Generator

**Mahesh S Naik**
Research Scholar
Shri Jagdishprasad Jhabarmal Tibrewala University, Rajasthan

*Abstract*— this paper is an outcome of approach towards designing a new random generator which follows basic properties like uniformity & independence. An attempt has been made to generate random numbers, which are non-predictable and avoid repetition of sequence.

***Key words:*** Pseudorandom Number, Uniformity, Independence, Simulation, Correlation

## I. INTRODUCTION

Random numbers play an important role in many applications like Monte Carlo simulation techniques, stochastic optimization, cryptography, physics, gaming, statistical mechanics etc. This application needs fast and reliable random number generation. Random numbers can be generated by using true processes, which are completely random but need proper instruments/devices for capturing and then conversion into digital format, which becomes a costly process. In many cases this may not be economically feasible. Thus in practice, the random numbers are produced by deterministic rules, which are based on arithmetic operations. Random no's produced by this methods/algorithms are finite and reproducible and not completely random. Less theoretical work exists on random number generation. Thus, the properties of many generators are not well understood. There has been a gradual increase in improved pseudorandom number generator algorithms, which has led to a diversity of generators available in computer software, public domain. Here are some uses for random numbers

- Simulation: Random numbers are necessary to make a realistic model of natural phenomena. These simulations include economic, nuclear physics and many other models.
- Sampling: a random sample would produce best possible accuracy of these statistical tests.
- Aesthetics: random numbers are used in music, art etc.
- Cryptography: Cryptography systems make use of large amounts of random data for generation of keys.
- Programming: computer algorithms use random number or random sequence; also random numbers can be used as input to test the effectiveness of an algorithm.
- Numerical Analysis: problems that are complex can be approximated by techniques relying on random numbers.
- Decision making in real life decisions, computer algorithms and game theory, randomness plays an important role.
- Applications of random numbers include gambling and computer games also.
- Random.org: RANDOM.ORG server offers true random numbers through Internet. The randomness comes from atmospheric noise, which in many cases is better than the pseudo-random number algorithms typically used in computer programs. RANDOM.ORG is used for holding drawings, lotteries and sweepstakes, to drive online games, for scientific applications and for art and music. Many other similar servers are available.

## II. TYPES OF RNG'S

There are two types, True random number generators (TRNG) and pseudorandom number generators (PRNG). True random number generators have no periodicities, no predictability, no dependencies, and a high level of security and are conceptually nice but they are slow, inefficient and costly, their sequences are not reproducible and they are subject to manipulation. A true random number generator requires a naturally occurring source of randomness, Pseudo random numbers are not strictly random, rather they are deterministic, meaning that they are computed using a mathematical algorithm. If the algorithm and the seed are known then the numbers generated are predictable. The main objective with PRNGs is to obtain sequences that behave as if they are random. The output sequences of many PRNGs are statistically indistinguishable from completely random sequences and ironically, PRNGs often appear to be more random than random numbers obtained from TRNGs. Following are some commonly used pseudorandom generator algorithms:

### *A. Linear Congruential generators (LCG's)*

Are calculated on the basis of four integers:

- m, the modulus, with m >0
- a, the multiplier, with 0 < a <m
- c, the increment, with 0 <c < m
- $x_0$, the seed, with 0< $x_0$< m

The sequence $[x_n]$ of random numbers is recursively defined as

$$x_{i+1} = (ax_i + c) \bmod m$$

if a number reappears in the sequence then the entire sequence begins to repeat.

*B. Lagged Fibonacci Generator*

A *lagged Fibonacci* generator requires an initial set of elements $X_1, X_2, \ldots, X_r$ and then uses the integer recursion

$$X_i = X_{i-r} \otimes X_{i-s}$$

Where r and s are two integer lags satisfying r > s and $\otimes$ is a binary operation (+, −,×, $\oplus$(exclusive-or)). The corresponding generators are designated by LF(r, s,$\otimes$).Generally, the initial elements are chosen as integers and the binary operation is addition modulo 2n.

*C. The Blum-Blum-Shub generator*

The Blum-Blum-Shub generator works as follows: Select two very large primes p; q both congruent to 3 mod 4 and compute the product m = pq. This m is the modulus. Start with seed s0 and compute the sequence s1; s2; s3; ::: by the formula

$$s_{n+1} = s_n^2 \bmod m$$

From this sequence, create a sequence of bits by

$$b_n = s_n \bmod 2$$

This is the desired pseudo random sequence.

*D. Mersenne Twister*

The Mersenne Twister is the most widely used general-purpose pseudorandom number generator. It was the first PRNG to provide fast generation of high-quality pseudorandom integers.

The most commonly used version of the Mersenne Twister algorithm is based on the Mersenne prime $219937-1$. The standard implementation of that, MT19937, uses a 32-bit word length. There is another implementation that uses a 64-bit word length, MT19937-64; it generates a different sequence.

other pseudorandom number generators are Inversive congruential generator, shift register generator, ISAAC (cipher), Maximal periodic reciprocals, Mersenne twister, Middle-square method, Multiply-with-carry, Naor-Reingold Pseudorandom Function, Park–Miller random number generator,RC4 PRGA, Well Equidistributed Long-period Linear etc.

## III. PROPOSED ALGORITHM FOR RANDOM NO GENERATION

With current generation of computing the drawbacks which come in technically have reduced. Thus the proposed algorithm uses basic ideas in programming to generate a random no. It takes some initial values and calculates the required random no. The calculation of the random variable $x_i$ is dependent on i (i is involved in calculation), the possibility of repetition of sequence reduces, because the $i^{th}$ term will occur only once.

Ideally random numbers produced should possess the following properties:

- The routine used to generate the numbers should be fast.
- Should have a sufficiently long cycle.
- The cycle length, or period, represents the length of the random number sequence until previously generated numbers begin to repeat.
- The random numbers should be reproducible. Being able to repeat the random numbers in the same order is very useful in comparing alternate systems.
- The generated random numbers should pass statistical tests for uniformity and independence.

Thus considering this attributes the following algorithm has been proposed,

*A. Proposed Algorithm:*

*1) Step No.1:*

Define the seed x0, multiplier p, & modulus m

*2) Step No.2:*

Obtain $x_{i-1}$/i

if the quotient is

<1 then multiply by 10000

<10 then multiply by 1000

<100 then multiply by 100

<1000 then multiply by 10

In this way a number with at least 4 digits will be generated before decimal, let the integral part of the same be denoted by n.

*3) Step No. 3:*

obtain k, k is sum of digits of n.

*4) Step No. 4:*

obtain l=(xi-1+i)*p

*5) Step No. 5:*

obtain xi=(k+l)%m

*6) Step No. 6:*

obtain xi/m

*7) Step No. 7:*
i=i+1
*8) Step No. 8:*
if I is less than required no. of random numbers goto step 2 else terminate the process.

*B. Advantages*

1) The no. generated using this steps are uniformly distributed & independent, which is the basic requirement of random numbers.
2) More over since sum of digits (two different no.s can yield the same sum of digits) is used & modulus is used, prediction becomes more difficult.
3) Also the i (the variable denoting $i^{th}$ random number) is used in calculation it brings variability and thus the sequence doesn't repeat even if the same no. occurs at a particular instance.
4) The variables and computation are not complex thus implementation in any language is possible in the simplest way.

*C. Disadvantages*

1) Algorithms yield good results for higher value of modulus m.
2) Seed value has to provide at the beginning of the algorithm.

*D. Implementation of the Algorithm in java*

```
public class Rng
{
   public static void main(String []args)
   {
      int x0=5,x1,p=11,m=223,k,l;
                    for(int i=1;i<=100;i++)
                    {   double x=(double)x0/i;
                       int n;
                       if(x<1)
                       n=(int)(x*10000);
                       else if(x<10)
                         n=(int)(x*1000);
                       else if(x<100)
                         n=(int)(x*100);
                       else if(x<1000)
                       n=(int)(x*10);
                       else
                       n=(int)x;
                       k=sumd(n);
                       l=(x0+i)*p;
                       x1=(k+l) % m;
                      double x1r=(double)x1/m;
                      System.out.print("  "+x1r);
                     x0=x1;
                    }
}
              static int sumd(int n)
                  {
                     int sum=0,remd;
                     while(n!=0)
                     {remd=n%10;
                     sum=sum+remd;
                     n=n/10;}
                     return sum;
                  }
}
```

Output for seed value 5, p =11 & modulus m=223

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.3183 | 0.6591 | 0.4573 | 0.2825 | 0.3946 | 0.7130 | 0.2421 | 0.1390 | 0.0403 | 0.9775 |
| 0.3811 | 0.8654 | 0.2376 | 0.4080 | 0.3094 | 0.2376 | 0.5067 | 0.5605 | 0.2197 | 0.4529 |
| 0.1121 | 0.3677 | 0.2645 | 0.1793 | 0.2376 | 0.9551 | 0.9775 | 0.2556 | 0.3363 | 0.2107 |
| 0.9058 | 0.5964 | 0.2197 | 0.1390 | 0.3811 | 0.0224 | 0.1165 | 0.2466 | 0.6636 | 0.3183 |
| 0.5784 | 0.4843 | 0.4887 | 0.6367 | 0.286 | 0.4887 | 0.7623 | 0.8116 | 0.4394 | 0.3721 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0.681 | 0.1345 | 0.1704 | 0.6143 | 0.5381 | 0.7219 | 0.8251 | 0.9955 | 0.9417 | 0.3542 |
| 0.9820 | 0.9192 | 0.2780 | 0.3497 | 0.0672 | 0.0538 | 0.9775 | 0.1524 | 0.1793 | 0.5022 |
| 0.1165 | 0.8834 | 0.4304 | 0.4708 | 0.9013 | 0.7354 | 0.9506 | 0.3811 | 0.1479 | 0.6278 |
| 0.6278 | 0.9820 | 0.9147 | 0.2376 | 0.8385 | 0.4349 | 0.0762 | 0.2152 | 0.7892 | 0.1793 |
| 0.4843 | 0.8878 | 0.3497 | 0.5515 | 0.7578 | 0.1255 | 0.1973 | 0.0358 | 0.2959 | 0.2466 |

Table 1: Output

## IV. TESTING OF RANDOM NUMBERS

Random numbers were generated using the above process and were tested using various test for desired properties as follows:

### *A. Testing for Uniformity*

Kolmogorov Smirnov & Chi Square test was performed to test the uniformity of random numbers generated. Both tests measure the agreement between the distribution of a sample of generated random numbers and the theoretical uniform distribution.

We have two hypotheses, one says the random number generator is indeed uniformly distributed. We call this $H_0$, known in statistics as *null hypothesis*. The other hypothesis says the random number generator is not uniformly distributed. We call this $H_1$, known in statistics as *alternative hypothesis*. We are interested in testing result of H0, reject it, or fail to reject it

### *B. Kolmogor smirnov test*

- $H_0$: $R_i \sim U[0,1]$
- $H_1$: $R_i \sim U[0,1]$

| j | r | j/n | (j-1)/n | j/n-r | r-(j-1)/n |
|---|---|---|---|---|---|
| 1 | 0.0403 | 0.05 | 0 | 0.0097 | 0.0403 |
| 2 | 0.139 | 0.1 | 0.05 | -0.039 | 0.089 |
| 3 | 0.2197 | 0.15 | 0.1 | -0.0697 | 0.1197 |
| 4 | 0.2376 | 0.2 | 0.15 | -0.0376 | 0.0876 |
| 5 | 0.2376 | 0.25 | 0.2 | 0.0124 | 0.0376 |
| 6 | 0.2421 | 0.3 | 0.25 | 0.0579 | -0.0079 |
| 7 | 0.2825 | 0.35 | 0.3 | 0.0675 | -0.0175 |
| 8 | 0.3094 | 0.4 | 0.35 | 0.0906 | -0.0406 |
| 9 | 0.3183 | 0.45 | 0.4 | 0.1317 | -0.0817 |
| 10 | 0.3811 | 0.5 | 0.45 | 0.1189 | -0.0689 |
| 11 | 0.3946 | 0.55 | 0.5 | 0.1554 | -0.1054 |
| 12 | 0.408 | 0.6 | 0.55 | 0.192 | -0.142 |
| 13 | 0.4529 | 0.65 | 0.6 | 0.1971 | -0.1471 |
| 14 | 0.4573 | 0.7 | 0.65 | 0.2427 | -0.1927 |
| 15 | 0.5067 | 0.75 | 0.7 | 0.2433 | -0.1933 |
| 16 | 0.5605 | 0.8 | 0.75 | 0.2395 | -0.1895 |
| 17 | 0.6591 | 0.85 | 0.8 | 0.1909 | -0.1409 |
| 18 | 0.713 | 0.9 | 0.85 | 0.187 | -0.137 |
| 19 | 0.8654 | 0.95 | 0.9 | 0.0846 | -0.0346 |
| 20 | 0.9775 | 1 | 0.95 | 0.0225 | 0.0275 |

Table 2: Test

| | |
|---|---|
| D=max | 0.2433 |
| adjD | 1.123251 |

| | | | | | |
|---|---|---|---|---|---|
| *alpha* | 0.150 | 0.100 | 0.050 | 0.025 | 0.010 |
| *critVal* | 1.138 | 1.224 | 1.358 | 1.480 | 1.628 |

Since adjD is less than all the critical values above

We fail to reject the null hypothesis, H0, means that evidence of non-uniformity has not been detected at all levels of significance as specified above.

### *C. Chi Square test*

- $H_0$: $R_i \sim U[0,1]$

– $H_1$: $R_i \sim U[0,1]$

| Class | Oi | Ei | (Oi-Ei)^2 | (Oi-Ei)^2/Ei |
|---|---|---|---|---|
| 1 | 6 | 10 | 16 | 1.6 |
| 2 | 14 | 10 | 16 | 1.6 |
| 3 | 17 | 10 | 49 | 4.9 |
| 4 | 13 | 10 | 9 | 0.9 |
| 5 | 11 | 10 | 1 | 0.1 |
| 6 | 7 | 10 | 9 | 0.9 |
| 7 | 7 | 10 | 9 | 0.9 |
| 8 | 6 | 10 | 16 | 1.6 |
| 9 | 6 | 10 | 16 | 1.6 |
| 10 | 13 | 10 | 9 | 0.9 |
| observed value = | | | | 15 |
| critical value = | | | | 16.919 |

Table 3: Test

Since observed value is less than all the critical value obtain at 5% level of significance,

We fail to reject the null hypothesis, H0, means that evidence of non-uniformity has not been detected at all levels of significance as specified above.

*D. Correlation Test*

Correlation coefficient gives a quantitative measure of some type of correlation and dependence between variables.If the correlation coefficient is close to 0, there is less correlation between the numbers. If it is close to one, there is a large positive correlation, and if it is close to negative one there is a large negative correlation.

| r1 | r2 | r3 | | |
|---|---|---|---|---|
| 0.3183 | 0.6591 | 0.4573 | | |
| 0.6591 | 0.4573 | 0.2825 | | |
| 0.4573 | 0.2825 | 0.3946 | | |
| 0.2825 | 0.3946 | 0.713 | **correlation between r1 & r2** | **0.130582** |
| 0.3946 | 0.713 | 0.2421 | **correlation between r2 & r3** | **0.1387043** |
| 0.713 | 0.2421 | 0.139 | **correlation between r1 & r3** | **0.018895** |
| 0.2421 | 0.139 | 0.0403 | Note: r1, r2 & r3 each consist of 98 random numbers generated from the above algorithm. | |
| 0.139 | 0.0403 | 0.9775 | | |
| 0.0403 | 0.9775 | 0.3811 | | |
| 0.9775 | 0.3811 | 0.8654 | | |
| 0.3811 | 0.8654 | 0.2376 | | |
| 0.8654 | 0.2376 | 0.408 | | |
| 0.2376 | 0.408 | 0.3094 | | |
| 0.408 | 0.3094 | 0.2376 | | |
| 0.3094 | 0.2376 | 0.5067 | | |
| 0.2376 | 0.5067 | 0.5605 | | |
| 0.5067 | 0.5605 | 0.2197 | | |
| 0.5605 | 0.2197 | 0.4529 | | |

Table 4: Test

The results of the calculation are shown in Table above.

Since the correlation between r1 and r2 is 0.1305, which is small, there is no indication of dependence between consecutive numbers.

Similarly, the correlation between r2 and r3 is
0.1387 & r1& r3 is 0.0188 which is also small. Thus it can be concluded that the numbers generated in this process are independent.

*E. Visual Test*

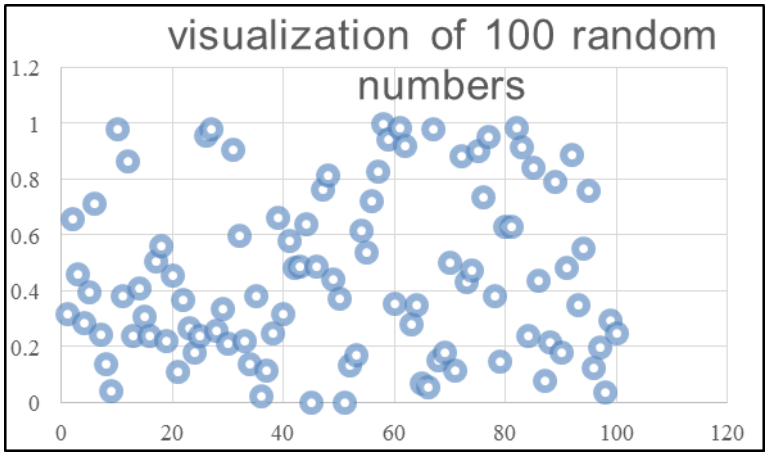


Fig. 1: Visual Test

100 random numbers generated through the algorithm above were plotted and do not show any pattern and also are scattered in the plane. Which provide an evidence of good sequence with required properties.

## V. CONCLUSION & FUTURE ENHANCEMENTS

This paper first analyses the available pseudo random number generators, and then gives a brief description of the proposed generator in terms of the algorithm. The paper also concludes that the proposed method of generation of random no's passes the test of uniformity & Independence. Also visual test adds to reliability of this method. In future this generator will be tested using diehard tests suite & NIST suite. Also the method cam be implemented and can be used in computer applications, programming languages & simulation.

## REFERENCES

[1] Park S K, Miller K.M.Random number generators: good ones are hard to find[J].New York: Communications of the ACM,1998, 31(10): 1192-1201.
[2] Haahr, Mads. random.org. 30 June 2000. <http://www.random.org>.
[3] The Distributed Computing Group, Trinity College Dublin. http://www.dsg.cs.tcd.ie/
[4] Foley, Louise. Analysis of an On-Line Random Number Generator, MSISS Project Report, April 2001.
[5] Brief Investigation into Random Number Generation http://people.bath.ac.uk/tdj20/gee.html#about
[6] Knuth, Donald E., The Art of Computer Programming – Seminumerical Algorithm. Vol 2 Chapter 3 Random Numbers pg1-184. 1997.
[7] G. Marsaglia and A. Zaman, Stat. & Prob. Lett. 8, 329 (1990).
[8] G. Marsaglia, B. Narasimhan and A. Zaman, Comp. Phys. Comm. 60,345 (1990).
[9] L'Ecuyer and Hellekalek, Random Number Generators: Selection Criteria and Testing, Lecture Notes in Statistics, New York Springer.
[10] P. L'Ecuyer, Random Numbers, in the International Encyclopedia of the Social and Behavioral Sciences, N. J. Smelser and Paul B. Baltes Eds., Pergamon, Oxford, 2002, 12735-12738.
[11] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 2nd ed. (Addison-Wesley, Reading, Massachusetts, 1981).